Remarks

In the application, claims 1 through 24 are pending. No claims currently stand allowed.

The Office Action dated September 30, 2003, has been carefully considered. The Office Action rejects claims 1 through 10 and 12 through 24 under 35 U.S.C. § 102(b) as anticipated by U.S. Patent 5,790,861 ("Rose"). Claim 11 is rejected under 35 U.S.C. § 103(a) as obvious in light of Rose and U.S. Patent 6,385,769 ("Lewallen").

Claims 7, 9, 11, and 22 are amended to address minor informalities. Claims 1, 18, and 22 are amended to more particularly point out and distinctly claim the subject matter of the present invention. Specifically, the phrase "during program execution" is added.

The present application and Rose both describe techniques for replacing a base software object class with another object class. However, the timing of this replacement differs, and that difference significantly affects the utility of these techniques.

Rose describes traditional base-class inheritance where the replacement *always occurs at compile or link time*. As a first example, see Rose's Figure 2 where the base and replacement objects are compiled and then linked (step 260). In the logic of Figure 2, a new replacement class definition requires a new compilation and link. The entire text of Rose's specification supports this view. For specific statements, see, e.g., the Abstract and column 3, lines 43 through 48 ("As a result, even if header file class definitions are subsequently modified, it will often be possible to generate updated executable code *by recompiling the modified header files and relinking the resulting object code*, without necessarily recompiling the application program code.") (emphasis added).

In contrast to Rose's compile- (or link-) time replacement, the present invention allows class replacement *during program execution*. This is true polymorphism and even works for a replacement class *that was unknown at compile time*, a feature clearly not disclosed by Rose. The present specification emphasizes this timing on page 14, lines 8 through 10 ("In contrast [with a Rose-type system], in the example given in FIG. 2, the replacement class B *may be designed after the reusable module 70 was generated.*"), page 14, lines 14 through 17 ("The class replacement in accordance with the invention enables *dynamic creation of objects of a new derived class* instead of objects of a base class *during execution of the existing code in the program.*"), page 15, line 24, through page 16, line 1 ("*During the execution of the program 94*, the new code in the module 90

calls special instructions to indicate, or register, the class replacement relationship between the classes A and B in a particular context."), page 30, line 24 through 27 ("In contrast, the automatic class replacement is carried out by *dynamically* passing the correct CLSREF value for the class of the object to be created."), and page 38, line 38, through page 39, line 6 ("Although in this example the Create function is invoked in the 'main' portion of the program, it will be appreciated that *the instruction to create objects of the replaceable classes may be included in an existing reusable module that was developed before the replacement classes were designed.* Note that the old code needs to know only about the CobjCreator object and the replaceable base classes. *It did not have to know about the new derived classes at compile time* in order to be reusable for creating objects of the new classes.") (emphasis added).

The present invention, unlike Rose, also allows the replacement to be undone during program execution. See, e.g., page 16, lines 5 through 7 ("Later, the new code may 'un-register' the class replacement so that a subsequent call to the old code will result in the creation of objects of the class A as usual."), and page 31, line 28, through page 32, line 2 ("In a preferred embodiment, another compiler-supported function is provided for un-registering the class replacement, thereby allowing class replacement to be dynamically switched on and off during program execution.")

This timing distinction over Rose is included in every pending independent claim, as currently amended. For example:

Claim 8. A computer-readable medium having computer programming source code comprising:

a first portion of the source code having:

. . .

a declaration that the base class *is replaceable during program execution*;

. . .; and

a second portion of the source code having:

. . .

an instruction to replace the base class with the replacement class during program execution.

(Emphasis added.)

Rose simply does not teach execution-time class replacement. As Rose neither anticipates nor renders obvious this feature of every pending claim, applicants request that the rejections be withdrawn and that all currently pending claims be allowed.

Conclusion

The application is considered in good and proper form for allowance, and the Examiner is respectfully requested to pass this application to issue. If, in the opinion of the Examiner, a telephone conference would expedite the prosecution of the subject application, the Examiner is invited to call the undersigned attorney.

Respectfully submitted,

John T. Bretscher, Reg. No. 52,651
One of the Attorneys for Applicants
LEYDIG, VOIT & MAYER, LTD.
Two Prudential Plaza, Suite 4900
180 North Stetson
Chicago, Illinois 60601-6780
(312)616-5600 (telephone)
(312)616-5700 (facsimile)

Date: March 5, 2004